|  |
| --- |
| **Functional Specification Document**  **EventHub** |

**DOCUMENT VERSION 1.2**

**04/19/2024**

**AUTHORS**

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Department** |
|  |  |  |
|  |  |  |

**DOCUMENT HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Document Revision Description** | **Document Author** |
| 17/04/2024 | 1.1 | Adding details to the documents |  |
| 19/04/2024 | 1.2 | Making minor changes |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**APPROVALS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Approval Date** | **Approved Version** | **Approver Role** | **Approver** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[1. Introduction 3](#_heading=h.30j0zll)

[1.1 Purpose of the document 4](#_heading=h.3znysh7)

[1.2 Project Scope 4](#_heading=h.2et92p0)

[1.3 Scope of the document 4](#_heading=h.tyjcwt)

[1.4 Related documents 4](#_heading=h.3dy6vkm)

[1.5 Terms/Acronyms and Definitions 4](#_heading=h.1t3h5sf)

[1.6 Risks and Assumptions 4](#_heading=h.4d34og8)

[2. System/ Solution Overview 5](#_heading=h.2s8eyo1)

[2.1 Context Diagram/ Interface Diagram/ Data Flow Diagram, Application Screen Flow, Sitemap, Process Flow 5](#_heading=h.3rdcrjn)

[2.2 System Actors 5](#_heading=h.lnxbz9)

[2.3 Dependencies and Change Impacts 5](#_heading=h.44sinio)

[3. Functional Specifications 5](#_heading=h.2jxsxqh)

[3.1 <Title> 6](#_heading=h.z337ya)

[4. System Configurations 8](#_heading=h.3j2qqm3)

[5. Other System Requirements/ Non-Functional Requirements 9](#_heading=h.1y810tw)

[6. Reporting Requirements 9](#_heading=h.4i7ojhp)

[7. Integration Requirements 9](#_heading=h.2xcytpi)

[7.1 Exception Handling/ Error Reporting 9](#_heading=h.1ci93xb)

[8. Data Migration/ Conversion Requirements 10](#_heading=h.3whwml4)

[8.1 Data Conversion Strategy 10](#_heading=h.2bn6wsx)

[8.2 Data Conversion Preparation 10](#_heading=h.qsh70q)

[8.3 Data Conversion Specifications 10](#_heading=h.3as4poj)

[9. References 10](#_heading=h.1pxezwc)

[10. Open Issues 10](#_heading=h.49x2ik5)

[Appendix 10](#_heading=h.2p2csry)

# Introduction

This documentation aims to address the business need for a chat web server. In today's digital age, real-time communication is essential for various applications, such as social networking platforms, customer support systems, and collaborative work environments. A chat web server provides the infrastructure and functionality to enable users to engage in instant messaging, group chats

## Purpose of the document

This document provides a detailed description of how the chat web server will work and what it will do. It serves as a guide for developers and stakeholders, outlining the specific requirements, features, and functionality of the system. The document also helps ensure that the chat web server aligns with the initial business goals and requirements. It includes use cases, diagrams, and mock-ups to provide a clear understanding of how the system will function. Overall, the document acts as a roadmap for the development and implementation of the chat web server.

## Project Scope

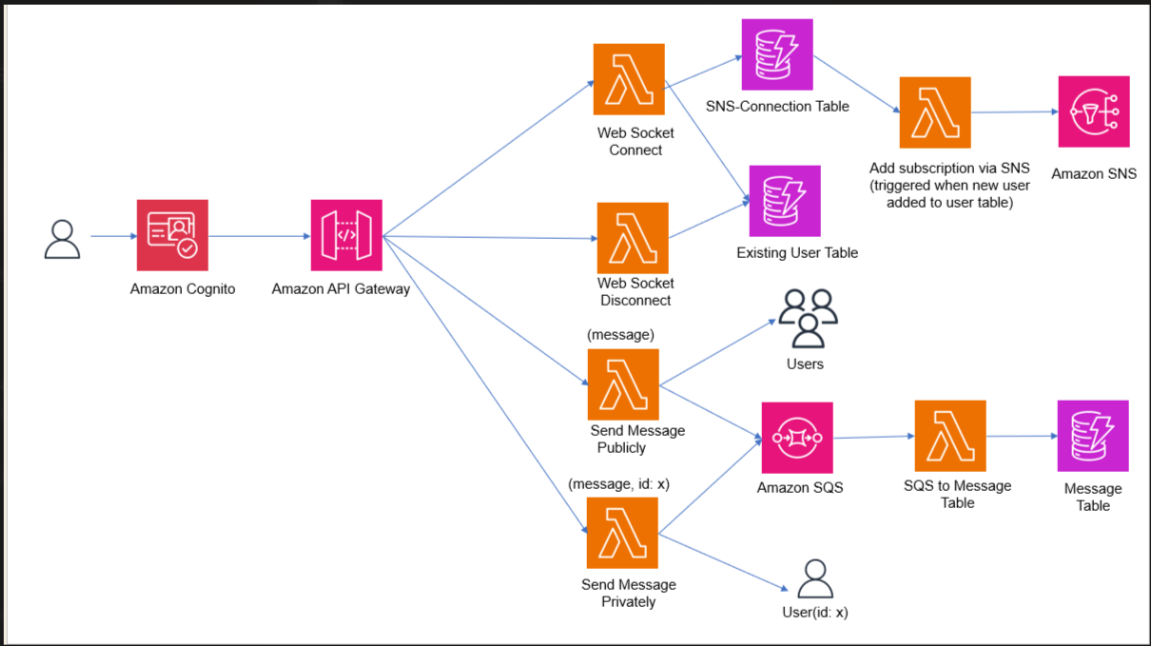
## 1.3 Risks and Assumptions

Data Breaches and Unexpected Cost Overruns are two specific risks associated with developing a chat web server with an AWS backend. Data breaches pose a threat to user privacy and can result in reputational damage and legal consequences. Inadequate security measures or misconfigurations in the AWS backend can expose user data to unauthorized access. However, unexpected cost overruns can occur if effective cost management strategies are not in place. This risk can lead to financial strain, project delays, or compromises on system features or performance. Mitigating these risks requires implementing robust security measures and regularly monitoring and optimizing AWS resources to control costs effectively.

# System/ Solution Overview

The software being specified is a web-based chat application that utilizes Node.js and TypeScript for the frontend, and AWS services for the backend. The relevant benefits of this solution include seamless and efficient messaging, secure user authentication and authorization through AWS Cognito, and the ability to scale and handle high user loads using AWS WebSocket API Gateway. The use of AWS services such as DynamoDB and SQS enables reliable message storage and processing, while AWS SNS allows for email notifications to be sent when a new user logs in.

## 2.1 Context Diagram/ Interface Diagram/ Data Flow Diagram, Application Screen Flow, Sitemap, Process Flow



## 

## 2.2 System Actors

### 2.2.1 User Roles and Responsibilities / Authority Requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User/Role** | **Example** | **Frequency of Use** | **Security/Access, Features Used** | **Additional Notes** |
| Chat App User | User 1 | Frequent | The Chat App User has access to the chat functionality of the application, including sending and receiving messages, participating in group chats, and accessing message history. They have the authority to create and manage their own chat groups and interact with other users. The user's access is authenticated and authorized through AWS Cognito, ensuring secure communication within the app. | N/A |

## 2.3 Dependencies and Change Impacts

### 2.3.1 System Dependencies

The proposed solution for the chat app has dependencies on various systems. It relies on AWS services such as WebSocket API Gateway, DynamoDB, SQS, SNS, and Cognito for its functionality and infrastructure.

### 2.3.2 Change Impacts

Changes may affect how users interact with the app, its speed and responsiveness, compatibility with different devices, security measures, ability to handle increased traffic, and integration with other systems. Careful consideration and thorough testing are necessary before implementing any changes to ensure a smooth and successful transition.

# Functional Specifications

## The overall system specifications for the chat app include user authentication and authorization, real-time messaging, message history and search, notifications, file sharing, user interface and user experience, scalability and performance, security and privacy, integration with AWS services, and monitoring and logging capabilities. These specifications ensure a secure, scalable, and user-friendly chat app that meets the business requirements and functional needs. Test cases will be created to validate these specifications and ensure the successful implementation of the system.

## 3.1 Chat Application

### 3.1.1 Purpose/ Description

The purpose of a chat application is to facilitate real-time communication between individuals or groups. It allows users to send and receive text messages, share files, and sometimes make voice or video calls. Chat applications are commonly used for personal communication, such as staying in touch with friends and family, as well as for professional communication, such as team collaboration in the workplace. They provide a convenient and efficient way to communicate and exchange information in a fast-paced digital world.

### 3.1.2 Use case

|  |  |
| --- | --- |
| **UC-1** | **Chat Application** |
| **Primary Actor(s)** | Chat User |
| **Stakeholders and Interest** | Host |
| **Trigger** | When the user connections to the application |
| **Pre-conditions** | The user should have logged in |
| **Post-conditions** | The user should disconnect the chat |
| **Main Success Scenario** | 1. Login the application 2. Connect 3. Chat 4. Disconnect |
| **Extensions** | N/A |
| **Priority** | High |
| **Special Requirements** | N/A |
| **Open Questions** | N/A |

### 3.1.4 Functional Requirements

The chat app allows users to connect to the chat server, disconnect from the server, send messages privately to specific users, and send messages publicly to a chat room. The app includes user authentication to ensure secure access to the chat functionality. Users can log in with their credentials and have their identity verified before accessing the chat features.

### 3.1.5 Field level specifications

**Form Business Rules and Dependencies:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Field Label** | **Validation / Business Rules** | **Error Messages** | **Data Dependencies** | **Additional Info/ Notes** |
| User Mail | The user is signed up using valid email and validated using code sent on their mail. | If verification fails, display “user not verified” | None | After successful validation user can setup their new password and are all set to sign-in |

**Buttons, Links and Icons:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Button, Link, Icon Label** | **OnClick Event** | **Other Event** | **Visible** | **Enabled Vs Disabled** | **Navigate To** | **Validation** | **Dependencies** |
| Connect | WebSocket connection is set. | User is added to the user table. | After the connection is set, the button is not visible | Enabled | Chat Room | NA | NA |
| Disconnect | WebSocket connection is disabled. | User is removed from the user table. | After the connection is disabled, the button is not visible. | Enabled | Homepage | NA | NA |
| Send Message Private | Message is sent to the specified user | The user against whose id, message is sent, receives the message | Yes, always. | Enabled | Receiver’s user id field is prompted. | NA | NA |
| Send Message Public | Message is sent to all users. | Each user receives the message | Yes, always. | Enabled | Chat Room | NA | NA |

# System Configurations

# AWS Lambda:

* + Create a Lambda function to handle the business logic of your chat application, such as sending and receiving messages.
  + Configure the Lambda function to trigger based on events from other AWS services, such as DynamoDB or SQS.

1. Amazon DynamoDB:
   * Create a DynamoDB table to store chat messages, user information, and other relevant data.
   * Define the necessary attributes and indexes for efficient querying and retrieval of chat messages.
2. Amazon Cognito:
   * Set up a Cognito User Pool to manage user authentication and authorization.
   * Configure the user pool to allow users to sign up, sign in, and manage their account credentials securely.
3. WebSocket API:
   * Create an API Gateway WebSocket API to handle real-time communication between clients and the server.
   * Configure the API to integrate with your Lambda function for processing incoming messages and sending responses.
4. Amazon SQS (Simple Queue Service):
   * Set up an SQS queue to handle asynchronous processing of chat messages or other tasks that require decoupling.
   * Configure your Lambda function to consume messages from the SQS queue and process them accordingly.
5. Amazon SNS (Simple Notification Service):
   * Configure SNS to send notifications to subscribed users when new chat messages or other relevant events occur.
   * Integrate SNS with your Lambda function to trigger notifications based on specific conditions or events.

# Other System Requirements/ Non-Functional Requirements

1. Performance: The chat application should handle many concurrent users and messages without significant delays or performance degradation. It should have low latency and respond quickly to user actions.
2. Scalability: The application should be designed to scale horizontally, allowing for easy addition of more servers or resources to handle increased user load. It should be able to handle sudden spikes in traffic without impacting performance.
3. Reliability: The chat application should be highly reliable and available, ensuring that users can access and use it without interruptions. It should have mechanisms in place to handle failures, such as server crashes or network outages, and recover gracefully.
4. Security: Chat applications should prioritize the security and privacy of user data. This includes secure user authentication and authorization mechanisms, encryption of messages in transit and at rest, protection against common security threats like cross-site scripting (XSS) and SQL injection, and compliance with relevant data protection regulations.
5. Usability: The application should have a user-friendly interface that is intuitive and easy to navigate. It should provide features like message search, message history, and notifications to enhance the user experience.
6. Compatibility: The chat application should be compatible with various devices and platforms, including web browsers, mobile devices, and desktop applications. It should support different operating systems and browsers to ensure a wide user base.
7. Maintainability: The application should be designed and developed in a way that allows for easy maintenance and updates. It should have clean and modular code, proper documentation, and version control to facilitate future enhancements and bug fixes.
8. Integration: The chat application should be able to integrate with other systems or services, such as user management systems, databases, or third-party APIs, to provide additional functionality or data exchange.

# Integration Requirements

1. DynamoDB: The chat application web server needs to integrate with DynamoDB to store and retrieve chat messages, user information, and other relevant data.
2. SQS (Simple Queue Service): SQS can be used for asynchronous communication between different components of the application, such as sending notifications or processing background tasks.
3. SNS (Simple Notification Service): SNS can be used for sending real-time notifications to users, such as new message alerts or system updates.

## Exception Handling/ Error Reporting

|  |  |  |  |
| --- | --- | --- | --- |
| **Exception/ Error ID** | **Error** | **Cause** | **Solution Strategy** |
| Internal Server Error  Internal Server Error | Invalid Parameter Error  Connection Id not found | Entering wrong email when connection is being set  When user enters the wrong connection id while sending private message | Applied Regex  Send alert |

# References

**AWS Documentation:**

* [Using Lambda with Amazon SQS - AWS Lambda](https://docs.aws.amazon.com/lambda/latest/dg/with-sqs.html)
* [Tutorial: Building a serverless chat app with a WebSocket API, Lambda and DynamoDB - Amazon API Gateway](https://docs.aws.amazon.com/apigateway/latest/developerguide/websocket-api-chat-app.html)
* [Using AWS Lambda with Amazon DynamoDB - AWS Lambda](https://docs.aws.amazon.com/lambda/latest/dg/with-ddb.html)
* [Using Lambda with Amazon SQS - AWS Lambda](https://docs.aws.amazon.com/lambda/latest/dg/with-sqs.html)
* [Using AWS Lambda with Amazon SNS - AWS Lambda](https://docs.aws.amazon.com/lambda/latest/dg/with-sns.html)
* [Amazon DynamoDB Documentation](https://docs.aws.amazon.com/dynamodb/)
* [Working with WebSocket APIs - Amazon API Gateway](https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-websocket-api.html)
* [Amazon Cognito Documentation](https://docs.aws.amazon.com/cognito/)
* [Amazon Simple Queue Service Documentation](https://docs.aws.amazon.com/sqs/)
* [Amazon Simple Notification Service Documentation](https://docs.aws.amazon.com/sns/)
* [AWS Lambda Documentation (amazon.com)](https://docs.aws.amazon.com/lambda/)

**Chat PwC:**

https://chat.pwc.com/

# Appendix

1. Architecture Overview:
   * The architecture of the Chat App follows a client-server model, with the client built using Node.js and TypeScript, and the server utilizing various AWS services.
   * The high-level architecture diagram illustrates the components, including Cognito for user authentication, WebSocket API Gateway for real-time messaging, Lambda functions for message processing, SQS for message buffering, SNS for email notifications, and DynamoDB for message storage.
2. AWS Services Used:
   * Cognito: Used for user authentication and authorization, ensuring secure access to the chat platform.
   * WebSocket API Gateway: Facilitates real-time communication between users, enabling instant messaging capabilities.
   * Lambda functions: Handle the backend logic, processing and storing messages in the DynamoDB table.
   * SQS: Acts as a buffer for incoming messages, ensuring reliable message delivery and decoupling message processing.
   * SNS: Sends automated email notifications to existing users when new users are added to the platform.
   * DynamoDB: A cloud-based database used for secure and scalable storage of chat messages.
3. Implementation Details:
   * Step-by-step instructions for setting up and configuring each AWS service, including Cognito, WebSocket API Gateway, Lambda functions, SQS, SNS, and DynamoDB.
   * Code snippets and explanations for key implementation aspects, such as user authentication, WebSocket API Gateway configuration, Lambda function development for message processing, SQS integration for message buffering, and DynamoDB data storage.
4. Frontend Development:
   * Description of the front-end development approach using Node.js and TypeScript.
   * Explanation of the front-end components, including private messaging, broadcasting, and user notifications.
   * Code snippets and examples of frontend implementation for better understanding.
5. Security Considerations:
   * Overview of the security measures implemented in the chat app, such as user authentication with Cognito, data encryption, and secure communication over WebSocket API Gateway.
   * Best practices for ensuring data security and privacy within the chat app.
6. Scalability and Performance:
   * Discussion on the scalability and performance considerations considered during the chat app's development.
   * Explanation of how AWS services like Lambda and SQS contribute to the scalability and performance of the application.
   * Recommendations for optimizing performance and handling increased user loads.
7. Testing and Deployment:
   * Overview of the testing approach, including unit testing, integration testing, and load testing.
   * Deployment instructions and best practices for deploying the chat app on AWS, including considerations for different environments (development, staging, production).
8. Troubleshooting and Support:
   * Troubleshooting guide for common issues that may arise during the setup or operation of the chat app.
   * Contact information for support and assistance, including relevant documentation and resources.